COMPUTER SCIENCE TECHNICAL REPORT SERIES

UNIVERSITY OF MARYLAND
COLLEGE PARK, MARYLAND
20742

LEVEL II

1

TR-961
DAAG-53-76C-0138

October, 1980

TWO HIERARCHICAL LINEAR FEATURE REPRESENTATIONS: EDGE PYRAMIDS AND EDGE QUADTREES

Michael Shneier
Computer Vision Laboratory
University of Maryland
College Park, MD 20742

ABSTRACT

Two related methods for the hierarchical representation of curve information are presented. First, edge pyramids are defined and discussed. An edge pyramid is a sequence of successivel· lower resolution images, each image containing a summary of the edge or curve information in its predecessor. This summary includes the average magnitude and direction in a neighborhood of the preceding image, as well as an intercept in that neighborhood and a measure of the error in the direction estimate. An edge quadtree is a variable-resolution representation of the linear information in the image. It is constructed by recursively splitting the image into quadrants based on magnitude, direction and intercept information. Advantages of the edge quadtree representation are its ability to represent several linear features in a single tree, its registration with the original image, and its ability to perform many common operations efficiently.

The support of the U.S. Army Night Vision Laboratory and the Defense Advanced Research Projects Agency under Contract DAAG-53-76C-0138 (DARPA Order 3206) is gratefully acknowledged. The author would like to thank Ann Scher for producing the diagrams.

1. Introduction

Edges provide much information about the contents of an image. Often this information is hard to interpret because of the large amounts of data involved and the existence of spurious edges that arise from noise in the image. This paper presents two related approaches to representing edges that attempt to overcome some of the difficulties in analyzing edges. The first approach is based on the use of a pyramid, or sequence of images, each a lower resolution version of its predecessor. The second involves a variable resolution representation, in which the local consistency of the edges determines the resolution at which they are represented. This approach builds a quadtree from the image, with leaves in the tree storing information about the edges that pass through square subregions of the image. Both of the representations are able to represent not only edges, but any linear information.

Many researchers have taken advantage of the pyramid structure to devise various efficient image-processing algorithms ([8], [14], [30], [31], [32]) Most of these algorithms, however, have dealt with images containing extended homogeneous regions, or blobs. The use of pyramids in linear feature analysis is significantly more complex than in region analysis. This is because a pyramid is well suited for representing images whose major features are two dimensional. Such features tend to retain their integrity and remain recognizable when lower resolution versions of the image are constructed using a simple rule such as averaging over small local neighborhoods. In contrast, the important features of edge or curve images are concentrated in a small proportion of the image, and it is the positions and orientations of the edges or curves that are the important information in the image. This paper provides a method of constructing edge pyramids that allows the advantages of the pyramid structure to be applied to edge and curve images. Some of these advantages include the compression of data to manageable size, and the ability to direct costly analysis in small regions of the original image, or set parameters such as thresholds. Projecting down from a given level in the pyramid also gives rise to an image in which all the features are of a known minimum size, and which has had much of the noise smoothed out.

A system has been implemented that builds pyramids from edge images, and can reconstruct edge images from low resolution levels in the pyramid. This system includes an edge enhancement scheme that is interesting in its own right. It also shows empirically the ability of an edge pyramid to retain most of the useful information at low resolutions, and the ability to reduce the amount of noise in the image.

The edge quadtree representation uses the same information as the edge pyramid, but has the ability to change the resolution of the representation to account for the edge information in the image. Thus, where edges are long and have consistent directions, large portions of the edges can be represented by leaves high up in the quadtree. Where edges are close together, however, or at corners, much smaller leaves may be needed to represent the edge information. This gives rise to a polygonal approximation of the curve information. The quadtree is shown to be useful for representing several edges or curves in a single structure, in contrast to other representations like the strip trees of Ballard ([1]), the upright rectangles of Burton ([4]), and the chain codes of Freeman ([18]). The representation allows many common edge operations to be performed efficiently, and the fact that it is in registration with the image and with ordinary quadtree representations of region-like information built from the image, simplifies interaction between region and edge operations.

2. Edge Pyramids

2.1 Building an edge pyramid

A pyramid to be used in edge or curve analysis cannot simply be constructed by building an averaged pyramid and then applying an edge detector at each level. This is because smoothing in the pyramid might cause some edges to be missed, while those edges that are found will be displaced relative to the original image because the edge detector is operating on a different image. Were it not the case that direction information is crucial in edge analysis, a pyramid could be constructed from an edge magnitude image by using the maximum value of the magnitude in each neighborhood of the image as the value of a point in the successor level of the pyramid. Such an approach has been used to construct a pyramid of line information (Hanson & Riseman, [7])

For the purposes of this discussion it is assumed that an edge image has been constructed with information stored at the pixels through which the edges pass. In order to construct an edge pyramid that preserves as much information as possible between levels, certain information must be present at each point of each level, as a summary of the information in the corresponding neighborhood of its predecessor. In the implementation to be described below, the neighborhoods were of size 4 by 4, with overlap between adjacent neighborhoods. The minimum information to be stored at each point is an estimate of the magnitude and direction of the edge(s) through the corresponding neighborhood. In addition, an intercept point is needed to fix the position of an edge in a neighborhood. A measure that is also useful is an indication of the error in the direction estimate Errors will usually be high at corners or where more than one edge passes through a neighborhood. The error term can be used to signal such situations, and cause higher levels of the pyramid to ignore such regions. This gives rise to a class of edge quadtrees (Section 3).

Thus, it is necessary to provide a means of estimating the magnitude and direction of the edge(s) passing through a neighborhood, and an intercept point for each edge. To simplify matters, and to prevent the amount of information stored at each point from growing in an unbounded way, each neighborhood is restricted to having a single edge passing through it. Should more than one edge pass through a neighborhood, the best edge is sought using the following procedure.

The neighborhoods used in the implementation were of size 4 by 4, with each neighborhood sharing two rows with its vertical neighbors (North and South of it), and two columns with its horizontal neighbors (West and East). The neighborhoods are shown in Figure 1. The method is based on the observation that the central 2 by 2 regions are disjoint and cover the picture. Thus, by first finding the best path through the central regions, and then extending it to the full 4 by 4 neighborhoods, the complexity of computation is significantly reduced.

Each point in an edge image contains two pieces of information, a magnitude and a direction. Within the central 2 by 2 region of each 4 by 4 neighborhood, the points having the maximum magnitude, the next to maximum magnitude, and the minimum magnitude are found. Based on these values, a decision is made as to whether or not an edge exists in the neighborhood, and, if an edge exists, what kind of edge it is.

If the maximum value is greater than some minimum (currently 2 in the implementation) and the next to maximum value is greater than the minimum value, an edge with two points in the central region is assumed to exist unless the directions of the two points are not consistent (i.e. differ by more than 45 degrees). If one point is significantly greater than the rest in the central region, the assumption is made that an edge passes through the 4 by 4 neighborhood, but only touches one corner of the central 2 by 2 region. If no point differs significantly from its neighbors (i.e. by more than 2), no edge is assumed to pass through the neighborhood. Note that an edge could still pass through the 4 by 4 neighborhood. It would be represented, however, by the adjacent neighborhood through whose center it passed.

It must be understood that two kinds of direction information are used in evaluating edge continuation and edge consistency. First, there is the direction established by the edge detector as an estimate of the direction of the edge through a point. Second, there is the direction from a point in the grid of the image to another point. For example, a point has eight immediate grid neighbors, at angles of 0, 45, 90, ..., degrees around it. These fixed angles, together with the directions calculated for edges at a point and its neighbors, are used to establish the continuity and consistency of neighboring edge points.

An edge with two points passing through the central 2 by 2 region may consistently be extended in three ways at each end (Figure 2a). The assumption made is that edges do not change direction too radically (i.e. by more than 45 degrees per pixel). Thus, instead of looking for all possible sequences of four points through a 4 by 4 neighborhood, two sets of three continuations are all that need be examined. The directions of these points are required to be compatible with the two-point edge for them to be considered as eligible for continuing the edge. Should more than one extension be found at each end, the best is chosen based on the grid angle between the points, their directions, and their magnitudes. The best extension at each end is used to adjust the magnitude and direction of the corresponding central edge point.

For one-point edges there are five possible extensions in the 4 by 4 neighborhood (Figure 2b). The best compatible extension is used to adjust the edge point here too. (Note that although only two extensions are compatible with the assumption that edges bend gradually, the other three points must be examined to allow for the case where an edge terminates inside the neighborhood). If no compatible edge extension can be found, the edge magnitude is decreased.

The above process is applied to all 4 by 4 neighborhoods in parallel. It may be iterated to allow information to propagate along the edges. The result is a preferred edge through each neighborhood. These chosen edges are used to construct the edge pyramid.

The process that actually constructs the pyramid is much simpler than that which establishes the best edges. For each neighborhood, it must calculate a magnitude, direction, intercept, and direction error value. The magnitude is calculated as the mean of the magnitudes in the 4 by 4 neighborhood. The direction is calculated as the mean of the directions of those points in the neighborhood with non-zero magnitude. The error term is the square root of the sum of squares of differences between the individual directions and the mean direction. The intercept is one of four values, denoting the position of the maximum magnitude point in the 2 by 2 central region of the neighborhood. The values are only calculated for a region if an edge actually passes through the central 2 by 2 region. These values are sufficient to reconstruct the edge to within the error tolerance. Other, more complex pyramid construction methods could be implemented. For example, it would be possible to use information high up in the pyramid to alter decisions made earlier. Because a decision about the edge through a quadrant is made based only on local information, it might be found that a different decision would have made the edges higher up in the pyramid more consistent. By backtracking to lower levels and altering the decisions made there, perhaps a more informed result would be obtained.

Note that more than one edge might pass through a neighborhood.

In this case, it can be expected that the error term will be large, and the reconstruction less reliable. Reconstructing a level from its successor is also a simple process, although sophisticated and complex methods could be devised that would produce improved reconstructions. The method that was implemented makes no use of the error term, and does not require adjacent neighborhoods in the reconstructed image to be consistent. The process simply expands each point to a 4 by 4 neighborhood, and assigns the mean magnitude and direction stored at the point to each edge point in the expanded neighborhood. Points are chosen as edge points by requiring the edge to pass through the intercept point, and to lie along the assigned direction. The reconstructions that result are usually very reasonable, with the errors occurring only where there are sharp corners, or where edges are close together. The results can often be improved by applying the best edge routine discussed above.

2.2 Examples

The edge images for the examples presented below were obtained by applying a zero-crossing edge detector (Marr & Hildreth, [11]) to gray level images. The edge detector returns magnitude and direction values for points corresponding to zero crossings of the Laplacian or second directional derivative of the image intensity. The zero crossings are approximated by the zero points of the difference between two Gaussian-like functions with different standard deviations. The Laplacian is calculated using the hierarchical discrete correlation method of Burt ([3]). For each zero crossing point, a 5 by 5 Prewitt-like operator is used to give magnitude and direction information. The advantages of using a zero-crossing detector are that the edges are thin and the boundaries are closed curves. The edge pyramid process will, however, work for any edge or curve image.

The first example shows the entire process in a step by step way, using a binary image of a square. For most images, only the magnitude values produce meaningful pictorial displays, and only these images will be shown in later examples. In all the examples, the results of a 16-fold compression and a subsequent reconstruction are shown.

Figure 3a shows a binary image of a bright square of size 32 by 32 centered in a 64 by 64 image. Figure 3b shows the result of applying the zero-crossing edge detector to the image, while Figure 3c shows the image resulting from one iteration of the best edge procedure. In both cases, the top image is the direction image, and the bottom image is the magnitude image, both thresholded so that all non-zero points are displayed. After the best edge procedure has been applied, the directions of neighboring points are more consistent. This is the reason for the slight lengthening of the two short line segments at the bottom of the second direction image.

Figure 3d shows the four 32 by 32 images produced by the edge pyramid program. The images in the bottom row are the magnitude (left) and the intercept (right) of each point. Those in the top row are the direction image (left) and the error in direction (right). All images are thresholded so that non-zero points are displayed. The important thing to notice here is the error image. The only places at which errors in direction are detected are the corners.

Figure 3e shows the results of applying one iteration of the best edge procedure to the 32 by 32 images, and Figure 3f shows the 16 by 16 images produced by the pyramid process. The reconstruction algorithm is applied to the 16 by 16 images in Figure 3g, and to the reconstructed 32 by 32 images in Figure 3h. The results of the reconstruction illustrate the ability of the edge pyramid to retain full information in regions where there are long consistent lines. It is only at the corners that information is lost, and, in this case, a simple extension algorithm could be appplied to restore the corners.

Figure 4 shows the results of running the whole process on the edges produced from a gray level image of a tank (Figure 4a). The original edge magnitude and the enhanced magnitudes are shown in Figure 4b. Figure 4c shows the first level of the pyramid, the best edges found at this level, and the second level of the pyramid. The first level of reconstruction and the final reconstructed image are shown in Figure 4d. Figure 4e shows the result of thresholding the magnitude images, all at the same threshold value. It is clear that the important information has been retained. A similar sequence is shown in Figure 5, starting from a gray level image of part of an airport.

Figure 6, finally, shows what happens when an image has sharp corners and inconsistent edge that occur close together. While the result of running the process is still clearly recognizable, the edges have been broken up into very short segments at the corners, and the reconstructed image is of clearly inferior quality to the original.

3 Edge Quadtrees

A quadtree is obtained from a binary image by successive subdivision into quadrants. If the original image is homogeneous, a single leaf node is created. Otherwise, the image is divided into four quadrants, which become sons of the root node. This process is applied recursively until all terminal nodes are homogeneous. Binary quadtrees have been shown to be useful in representing large images compactly, and many algorithms have been developed for efficiently applying image processing techniques to images represented by quadtrees ([2], [5], [6], [9], [10], [19-29]).

For gray level images, a class of quadtrees can be defined, based on the brightness characteristics of the image. The root node represents the whole image, and typically stores the average gray level. If the image is sufficiently homogeneous (i.e. if the variance in gray level is not too great) no subdivision is performed. Otherwise, the image is divided into four subimages, and four children of the root are constructed. As long as the variance in any quadrant is higher than a threshold, the process is recursively applied. The result is a tree that represents the image to a degree of accuracy dependent on the threshold. By changing the threshold on the variance, a whole class of gray level quadtrees can be constructed. More generally, a class of quadtrees can be constructed using piecewise polynomial fits to the data in each quadrant. Gray level quadtrees are useful for image smoothing (Ranade & Shneier, [17]), shape approximation (Ranade <u>et al</u>., [15]), and segmentation (Ranade, [15], Wu <u>et al</u>.,

[33]).

Edge quadtrees are similar to the trees described above, except that the information stored at each node includes a magnitude, a direction, an intercept, and a directional error term. All the information about the edge that is stored is used in constructing the quadtree. As in the gray level quadtrees, a class of edge quadtrees can be defined based on the directional error term.

The construction of an edge quadtree proceeds by first examining the magnitude term, then the direction and direction error terms, and then the intercept term. If a node has a sufficiently low magnitude (i.e. no edge exists), then no subdivision is performed. Similarly, if the error term is sufficiently small, no subdivision is performed, with one exception, as follows. It may happen that a number of parallel edge segments run through a quadrant, so that the direction term is consistent with the data, and the error is low. Thus, a division based only on the error would not be performed. It is clear, however, that the quadrant does not represent the data at a sufficient level of detail to enable the set of parallel segments to be reconstructed. Thus, whenever the error term falls below threshold, a further check must be made to ensure that the intercept points in the quadrant are consistent (i.e. they lie along a line in the direction of the direction term). Should this not be the case, the quadrant must be subdivided. A final requirement for an edge quadtree is a flag that is turned on should an edge terminate within a quadrant. In this case, the intercept will be the point at which the edge terminates. The result of applying this process recursively to the image is a quadtree in which long edges that are nearly straight give rise to large leaves, or a succession of large leaves. Near corners, or where edges intersect, much smaller leaves are needed, perhaps as small as individual edge elements. A feature of edge images is the low percentage of points that contain interesting information. This means that an edge quadtree can be expected to contain many large leaves where there are no edges at all.

Figure 7 shows the edge magnitude image for the airplane picture of Figure 6, and the levels of the edge quadtree that are not empty (levels 0, 1, 2, and 3). Note that the leaves are upright square blocks in fixed positions, and that the shape of the quadtree is dependent on the global co-ordinate system of the image. This is a characteristic of all quadtrees. It is one of the major differences between this representation and the strip trees proposed by Ballard ([1]) (Section 4).

One of the advantages of the edge quadtree is that it can be used to represent an image that may contain more than one curve. Of course, a separate quadtree, perhaps smaller than the image quadtree, can be used to represent each curve, but it is preferable to use a single quadtree, both in order to maintain registration *with the image and for compactness. Each curve in the tree can be* named, and all terminal nodes representing part of a curve can be marked with the name of the curve. In addition, region-like information can be made available in the same structure, simplifying the interactions between regions and linear features. Notice that the quadtree for a closed curve and that for the region enclosed by the curve have closely related shapes.

If only one linear feature is represented by a quadtree, many operations become very efficient. If more than one curve is represented, however, some of the operations need to be done at a higher resolution, in order to ensure that the correct curves are involved. A way of alleviating this problem is to assign the names of curves passing through a quadrant to non-terminal as well as terminal nodes. A bound has to be put on the number of names allowed, however, because this may not be limited. All non-terminals that have more named curve segments passing through them than the bound allows can be flagged. When curve operations involving flagged nodes are executed, the descendants of the flagged nodes must be examined recursively to find the first one with the required names.

Many operations that are useful for manipulating edge and curve information can be implemented efficiently using edge quadtrees. Most of the algorithms are adaptations of quadtree algorithms for region representation. Only the broad outlines and necessary modifications will be given here.

First, an algorithm is presented for naming each curve in a quadtree. It is based directly on the algorithm for finding connected components of an image represented by a quadtree (Samet, [21]). First note that a leaf node can have no more than nine different curves passing through it . This is the number of "smooth" continuations through a sequence of three pixels, assuming less than a 90 degree change of angle between pixels. Thus the number of names at a leaf is bounded. (It is also necessary to assume that two or more curves cannot have arcs in common).

The algorithm involves three phases. The first pass assigns names to each curve node in the quadtree by starting at the North-West corner of the tree, and examining the South and East neighbors of the curve nodes. If the direction of a neighbor is compatible with node the (i.e. within the error tolerance) and its intercept is also compatible (i.e. lines up along the common direction with the node's intercept), the neighbor is given the same name. Otherwise, a new name is assigned. If a node is found that has already been named, and the node is compatible, an equivalence is established between the nodes.

The second phase processes the equivalent pairs to produce equivalence classes, while the third and final phase traverses the tree again, and assigns a single name to all members of an equivalence class. The names at the leaves of the tree can be propagated up to the non-terminal nodes at the same time, a non-terminal node being flagged if too many names are assigned to it

Many operations commonly applied to linear data are facilitated by the quadtree representation. For example, to find the length of a curve segment, the tree is traversed starting at the root, and looking at nodes until the first leaf node belonging to the segment is found. The length contributed by this node is calculated as the length of a line through the intercept with direction given by the direction component, and bounded by the node's borders. To find the rest of the nodes in the curve, the FIND-NEIGHBOR and FIND-CORNER algorithms defined by Samet([27]) are used. They are applied on each side in the direction given by the node's direction component. Nodes that are further away from the leaf's direction than is allowed by the error tolerance can be ignored. The lengths of the nodes found in this way are added to the curve length, if they have the correct name. Each new node after the first will have at most one successor. When no more neighbors can be found, the length has been calculated. For closed curves, the original node must be flagged to ensure that the process terminates.

Other operations are easily defined as modifications of regular quadtree operations. The distance from a point to a curve can be calculated using a variant of the distance transform algorithm (Samet, [25]). Instead of finding the distance from a point (or the center of a BLACK node) to the nearest WHITE, or boundary, point, the distance to the nearest curve point is found. Other algorithms, like union and intersection (Hunter & Steiglitz [9], Shneier [28]), require almost no alteration.

Interactions between region and boundary information are natural in this representation because of the registration of the images. Operations like the Superslice algorithm (Milgram [12]) can be performed on the quadtrees by taking advantage of the information contained in the shapes of the trees. The Superslice algorithm attempts to find the best segmentation of an image by matching edge and region information. A number of thresholds are applied to the image, giving rise to various new images. Each of these images is matched with the edge image, and that with the best region/boundary fit is chosen as the segmented image.

In the quadtree, this operation can be simplified and made more intelligent. Starting with the edge, and noting that the shape of the region quadtree is constrained by that of the edge quadtree, a class of candidate thresholds can be stored at each leaf node in the quadtree, each candidate giving rise to a region subtree with a shape consistent with the edge subtree. If, after all the nodes have had candidates assigned, there is a single threshold that gives the correct shape (i.e. the same threshold appears at all nodes), that threshold will give the required segmentation. Otherwise, a number of local thresholds may be applied to subimages, or an approximation to the segmentation can be made by choosing a compromise threshold.

4. Comparison with other methods

Another hierarchical quadrant-based method for representing edges is that of Omolayole and Klinger ([13]). They recursively subdivide an edge image into quadrants down to a 2 by 2 level. A number of edge patterns are then sought in each subquadrant, and, if too few of these are found, the quadrant is discarded. The result is a kind of tree structure, with the leaves containing template-like representations of the edge data in them. The main aim of this method seems to be to discard the areas of the image that contain little or no information. For edge images this can be expected to save fairly large amounts of storage. The edge quadtree differs from this approach in its treatment of quadrants containing edge *information*. Instead of having fixed-sized leaves, the quadtree allows leaves to be of the largest size consistent with the edge information they represent.

Other methods that have been devised for representing linear information are the upright rectangles of Burton ([4]), the strip trees of Ballard ([1]), and the chain codes of Freeman([18]).

Freeman has developed one of the most compact and well-known boundary representations, called chain codes. These codes represent the relative grid positions of successive line points in a digital image. They are perhaps not as well suited to representing edge information as line information, but have the advantages of being compact and are not tied to any particular co-ordinate system.

Burton presented a method of representing polygonal lines using a series of upright rectangles His work was extended by Ballard, who defined a representation for curve information called strip trees. A strip tree is a representation of a curve, obtained by successively approximating parts of the curve by enclosing rectangles. The structure is a binary tree, with the root node representing the bounding rectangle of the whole curve. This rectangle is broken into two parts at a point of maximum distance from the line joining the endpoints of the curve. The two parts are children of the root, and may be recursively subdivided until an error bound is satisfied. Note that the strip tree is not unique in cases where more than one extreme point exists.

All these other representations are able to represent only single curves, while the edge quadtree representation is able to represent several curves in the same tree. The edge quadtrees and edge pyramids are also in registration with the image, and with region-based representations like ordinary quadtrees and pyramids. This gives them a further advantage over the other representations. Where the other methods, and particularly the chain code, gain over the edge quadtree is in compactness, although the edge quadtree is actually storing more information than the other methods, and may give rise to better reconstruction of edge information.

5 Discussion

There are two main reasons for developing pyramids for linear features. The first is to provide compression of the data, for example to allow linear features to be detected from low resolution images in a hierarchical image data base, thus reducing the number of full resolution images that need to be examined. The second reason is to enable the most prominent edge features (or the edge features larger than a given size) to be extracted from the image, and to discard smaller features.

It would be impractical to search a large image data base for the existence of an object with a set of known features. Rather, it would be useful to be able to filter out most of the images on the basis of gross tests, leaving only a few to be examined more closely. For region- or blob-like features, this ability can be provided by gray-level pyramids or quadtrees. While the most natural form for storing linear information is probably a linked list, it is desirable for uniformity to store linear feature information in a similar way as regional feature information. This facility is provided by the edge pyramids and edge quadtrees presented in this paper. Of course the representations are useful not only for edges, but for any linear information

The second reason for building an edge pyramid is to enable noisy edges and edges that are too small to be filtered out of the image. In fact, this is achieved in two ways, both through the lateral best edge process and through the pyramid process. The best edge process is not designed specifically to enhance good edges and suppress bad ones, but it has this effect except where two edges intersect or two edges pass very close to each other The pyramid process causes short segments to be lost high in the pyramid because, after a while, they fail to find good continuations. Note, though, that short broken edge segments could be joined together if the gaps were sufficiently small relative to the image resolution. The main requirements for an edge to continue to exist at successively higher levels in the pyramid are consistency and good continuation.

6. Conclusions

Two related representations for linear information have been presented. Edge pyramids have been shown to be able to store the important edge information in an image, even at fairly low resolution, with the ability to reconstruct images that look very much like the originals. The main loss in information is at intersections of edges, or where edges pass close to each other. Small edges, usually representing noise, are also lost.

Edge quadtrees have been presented as an alternative hierarchical representation for linear feature information, with the ability to represent the information at variable resolution, depending on the local consistency of the data. The advantages of edge quadtrees over other representations are their ability to represent more than one curve in a single structure, and their registration with the image and with other region-based representations for images.

References

1. D. H. Ballard, Strip trees, a hierarchical representation for curves. Proc DARPA Image Understanding Workshop, SRI International, April 1979, 121-133.

2. P. J. Burt, Tree and pyramid structures for coding hexagonally sampled binary images. TR-814, Computer Science Center, University of Maryland, College Park, October 1979.

3. P. J. Burt, Fast, hierarchical correlations with Gaussian-like kernels. TR-860, Computer Science Center, University of Maryland, College Park, January 1980.

4. W. Burton, Representation of many-sided polygons and polygonal lines for rapid processing. CACM 20, 3, March 1977, 166-171.

5. C. R. Dyer, A. Rosenfeld, and H. Samet, Region representation: boundary codes from quadtrees. TR-732, Computer Science Center, University of Maryland, College Park, February 1979.

6. C. R. Dyer, Computing the Euler number of an image from its quadtree. TR-769, Computer Science Center, University of Maryland, College Park, May 1979.

7. A. R. Hanson and E. M. Riseman, Processing cones: a parallel computational structure for scene analysis. COINS working paper, University of Massachusetts at Amherst, 1976.

8. A. R. Hanson and E. M. Riseman, Segmentation of natural scenes. In Computer Vision Systems, (Hanson and Riseman, eds), Academic Press, New York, 1978.

9. G. M. Hunter and K. Steiglitz, Operations on images using quad trees. IEEE Trans PAMI-1, 2, April 1979, 145-153.

10. A. Klinger and C. R. Dyer, Experiments in picture representation using regular decomposition, Computer Graphics Image Processing 5, 1976, 68-105

11. D. Marr and E. Hildreth, Theory of edge detection. AI Memo 518, MIT AI Laboratory, April 1979.

12. D. L. Milgram, Region extraction using convergent evidence Computer Graphics Image Processing 11, 1979, 1-12.

13. J. O. Omolayole and A. Klinger, A hierarchical data structure scheme for storing pictures. In Pictorial Information Systems, (S. K. Chang and K. S. Fu, eds), Springer, 1980.

14. T. Pavlidis, Structural Pattern Recognition, Springer, 1977.

15. S. Ranade, A Rosenfeld, and H. Samet, Shape approximation using quadtrees. TR-847, Computer Science Center, University of Maryland, College Park, December 1979.

16. S. Ranade, A Rosenfeld, and J. M. S. Prewitt, Use of quadtrees for image segmentation. TR-878, Computer Science Center, University of Maryland, College Park, February 1980.

17. S. Ranade and M. Shneier, Using quadtrees to smooth images. TR-894, Computer Science Center, University of Maryland, College Park, April 1980.

18. A. Rosenfeld and A. C. Kak, Digital Picture Processing, Academic Press, New York, 1976.

19. H. Samet, Region representation: Quadtrees from boundary codes. TR-741, Computer Science Center, University of Maryland, College Park, March 1979.

20. H. Samet, Computing perimeters of images represented by quadtrees. TR-755, Computer Science Center, University of Maryland, College Park, April 1979.

21. H. Samet, Connected component labeling using quadtrees. TR-756, Computer Science Center, University of Maryland, College Park, April 1979.

22. H. Samet, Region representation: raster-to-quadtree conversion. TR-766, Computer Science Center, University of Maryland, College Park, May 1979.

23. H. Samet, Region representation: quadtrees from binary arrays. TR-767, Computer Science Center, University of Maryland, College Park, May 1979.

24. H. Samet, Region representation: quadtree-to-raster conversion. TR-768, Computer Science Center, University of Maryland, College Park, June 1979.

25. H. Samet, A distance transform for images represented by quadtrees. TR-780, Computer Science Center, University of Maryland, College Park, July 1979.

26. H. Samet, A quadtree medial axis transform. TR-803, Computer Science Center, University of Maryland, College Park, August 1979.

27. H. Samet, Neighbor finding techniques for images represented by quadtrees. TR-857, Computer Science Center, University of Maryland, College Park, January 1980.

28. M. Shneier, Linear time calculations of geometric properties using quadtrees. TR-770, Computer Science Center, University of Maryland, College Park, May 1979.

29. M. Shneier, Path length distance transforms for quadtrees TR-794, Computer Science Center, University of Maryland, College Park, July 1979.

30. M. Shneier, Using pyramids to define local thresholds for blob detection. Proc. DARPA Image Understanding Workshop, University of Southern California, November 1979, 31-35.

31. S. L. Tanimoto, Regular hierarchical image and processing structures in machine vision. In Computer Vision Systems (Hanson and Riseman, eds), Academic Press, New York, 1978.

32. L. Uhr, "Recognition cones", and some test results; the imminent arrival of well-structured parallel-serial computers; positions, and positions on positions. In Computer Vision Systems (Hanson and Riseman, eds), Academic Press, New York, 1978.

33. A. Y. Wu, T-H. Hong, and A Rosenfeld, Threshold selection using quadtrees, TR-886, Computer Science Center, University of Maryland, College Park, March 1980.

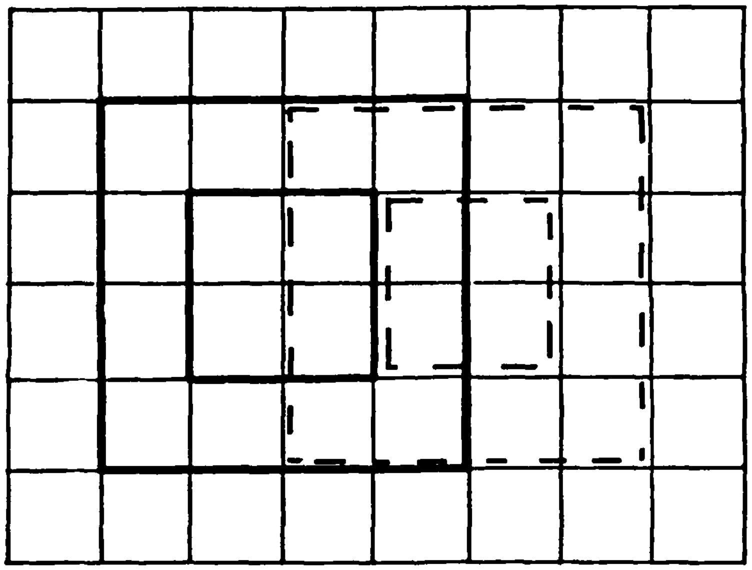
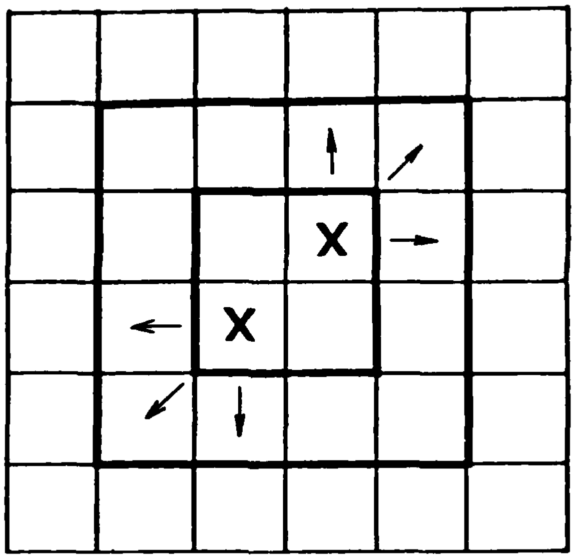
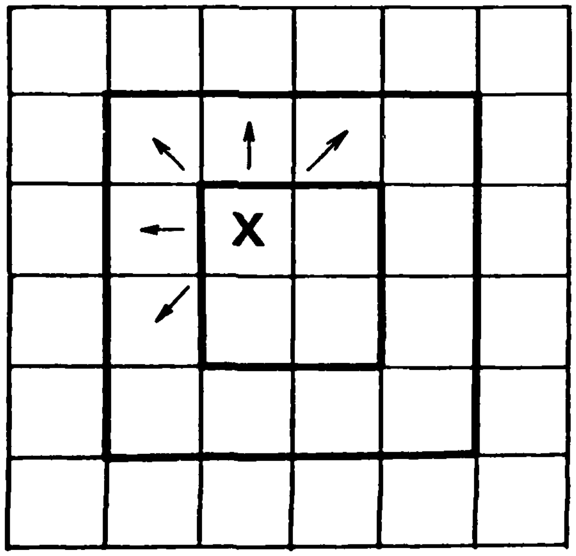


Figure 1. Two neighborhoods used in constructing pyramids. The central 2 by 2 regions are disjoint, but each neighborhood shares rows or columns with it neighbors.



(a)



(b)

Figure 2 The ways in which edge points may be extended. a. For a two-point edge, there are six consistent continuations. b. For a one-point edge, there are five consistent continuations.

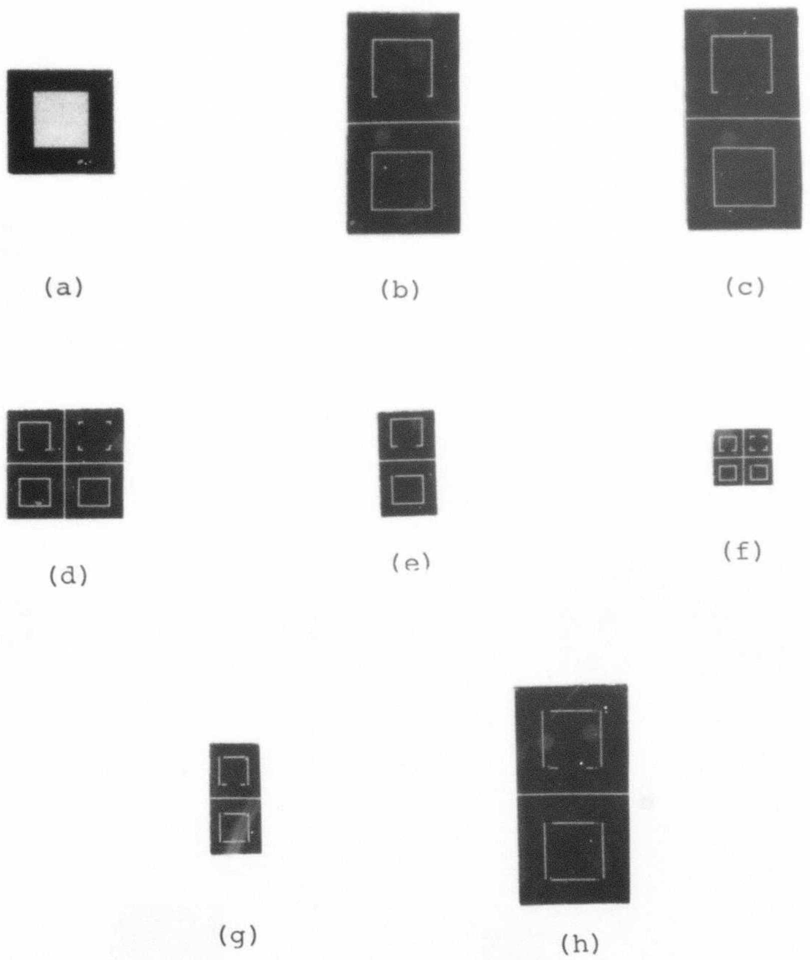
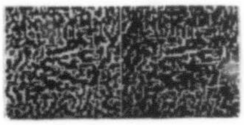
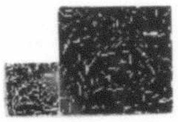


Figure 3. The pyramid process applied to a 64 by 64 binary image. a. Original image of a square region. b. The magnitude (bottom) and direction (top) images produced by a zero-crossng edge detector. c. The magnitude and direction images after running the best edge procedure. d. The first pyramid level (32 by 32): magnitude (bottom left); intercept (bottom right); direction (top left); error (top right). e. The result of applying the best edge procedure to the 32 by 32 images. f. The second pyramid level (16 by 16). g. The result of reconstructing a 32 by 32 image from the 16 by 16 pyramid level. h. The result of constructing a 64 by 64 image from the 32 by 32 reconstructed image.

(a)


(b)

(c)


(d)

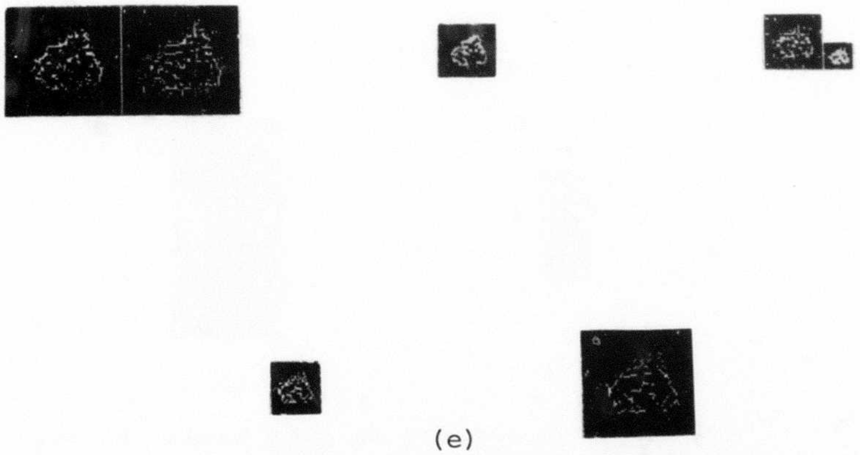
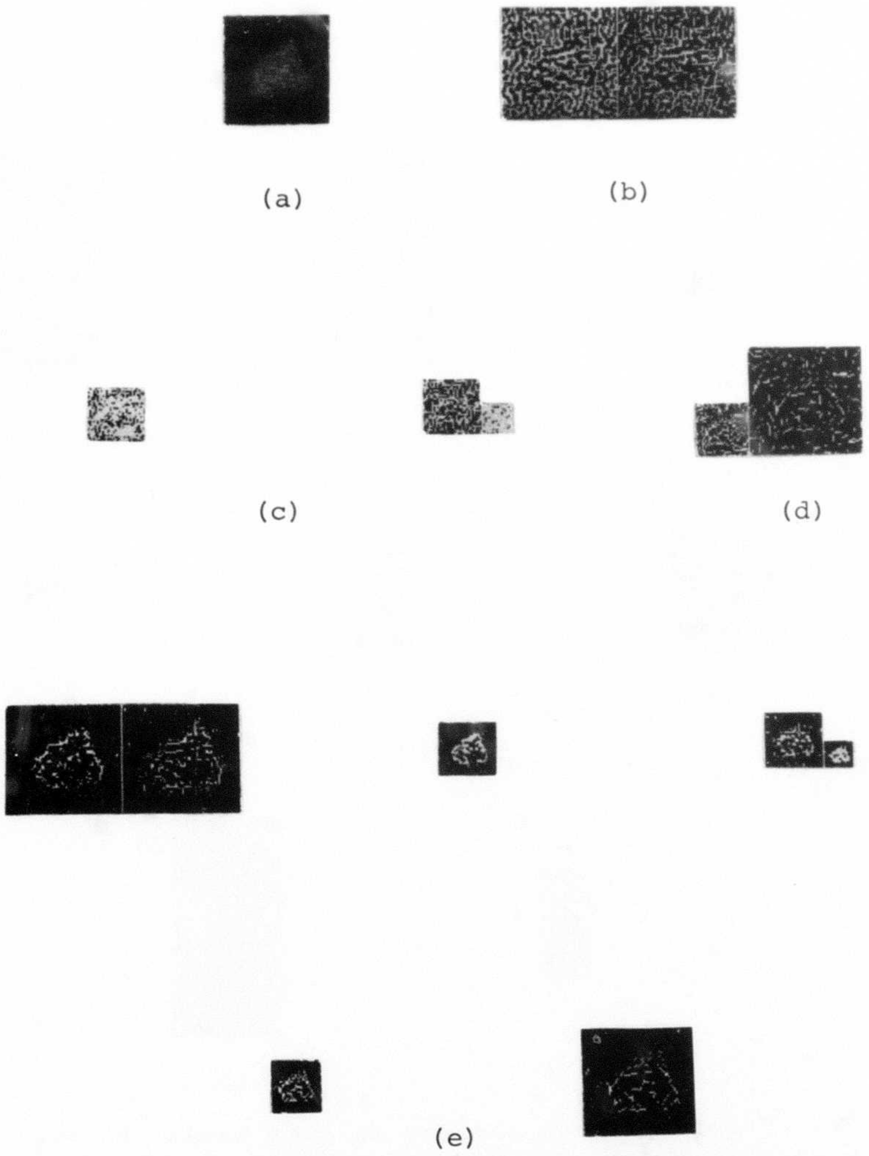

(e)

Figure 4. The pyramid process applied to a gray level image. a. A FLIR image of a tank. b. The magnitude and enhanced magnitude of the edge image of the tank (thresholded so that non-zero points are displayed). c. The first pyramid level magnitude, the enhanced magnitude, and the second pyramid level magnitude. d. The first and second level reconstructed images. e. The same process as in a and b above, but with all images thresholded at the same level.



<u>Figure</u> <u>4</u>. The pyramid process applied to a gray level image. a. A FLIR image of a tank. b. The magnitude and enhanced magnitude of the edge image of the tank (thresholded so that non-zero points are displayed). c. The first pyramid level magnitude, the enhanced magnitude, and the second pyramid level magnitude. d. The first and second level reconstructed images. e. The same process as in a and b above, but with all images thresholded at the same level.

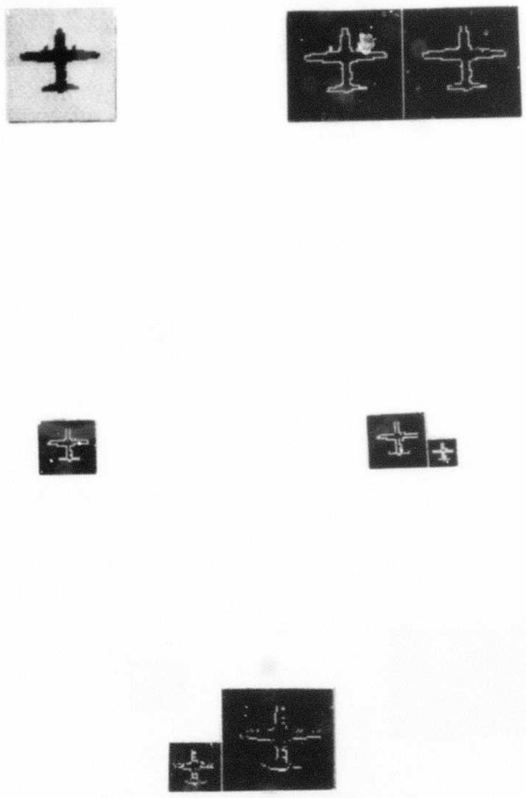
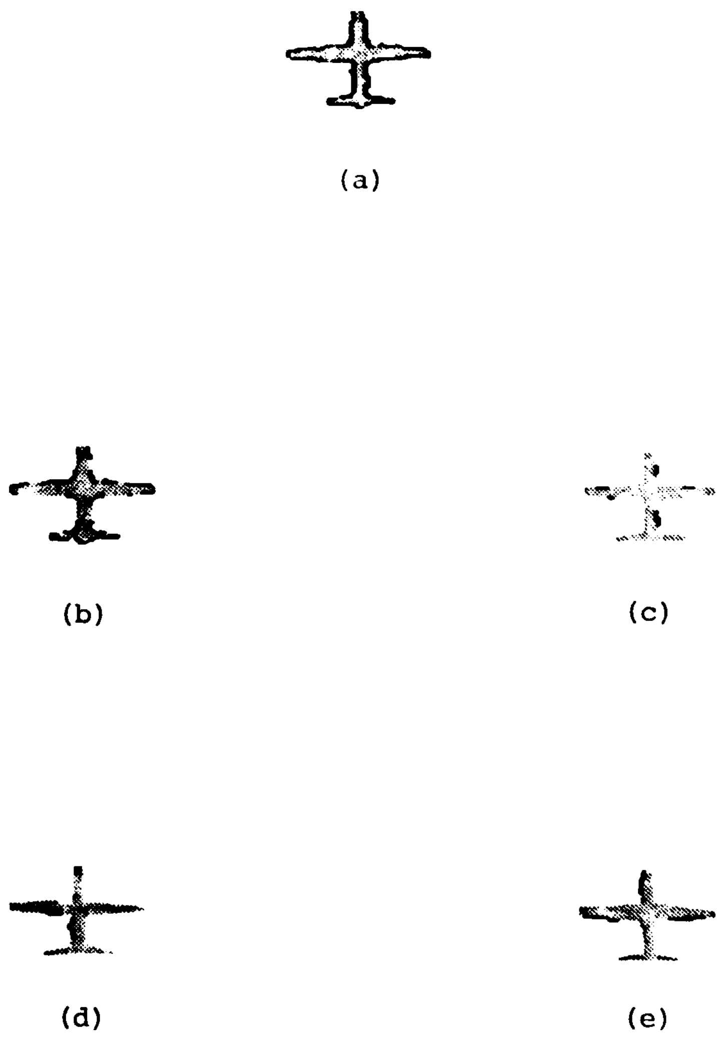


Figure 6. The pyramid and reconstruction process applied to a binary image of an airplane.



<u>Figure 7</u>. The edge quadtree of an airplane image. a. The edge magnitude image. b. The lowest level (level 0) of the edge quadtree (individual pixels). c. Level 1, having 2 by 2 blocks of pixels. d. Level 2, having 4 by 4 blocks. e. Level 3, having 8 by 8 blocks.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A109 561	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) TWO HIERARCHICAL LINEAR FEATURE REPRESENTATIONS: EDGE PYRAMIDS AND EDGE QUADTREES		5. TYPE OF REPORT & PERIOD COVERED Technical
		6. PERFORMING ORG. REPORT NUMBER TR-961
7. AUTHOR(s) Michael Shneier		8. CONTRACT OR GRANT NUMBER(s) DAAG-53-76C-0138
9. PERFORMING ORGANIZATION NAME AND ADDRESS Computer Vision Laboratory, Computer Science Center, University of Maryland, College Park, MD 20742		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS U. S. Army Night Vision Lab. Ft. Belvoir, VA 22060		12. REPORT DATE October, 1980
		13. NUMBER OF PAGES 35
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Image processing	Pyramids
Pattern recognition	Quadtrees
Linear features	
Hierarchical representations	

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Two related methods for the hierarchical representation of curve information are presented. First, edge pyramids are defined and discussed. An edge pyramid is a sequence of successively lower resolution images, each image containing a summary of the edge or curve information in its predecessor. This summary includes the average magnitude and direction in a neighborhood of the preceding image, as well as an intercept in that neighborhood and a

20. con't

measure of the error in the direction estimate. An edge quadtree is a variable-resolution representation of the linear information in the image. It is constructed by recursively splitting the image into quadrants based on magnitude, direction and intercept information. Advantages of the edge quadtree representation are its ability to represent several linear features in a single tree, its registration with the original image, and its ability to perform many common operations efficiently.